Universidad Simón Bolívar
Dpto. Computación y T.I
CI-3815, Organización del Computador
Sep-Dic 2017

Proyecto # 2 (10%)

Objetivo

Comprender los mecanismos para el procesamiento y tratamiento de excepciones e interrupciones en Mars.

Algunas definiciones

Tiempo Exclusivo

Es un sistema de trabajo de un computador gracias al cual se puede ejecutar un solo programa en un momento dado. Una vez que empieza a ejecutar un programa, éste continuará haciéndolo hasta su finalización o interrupción.

Tiempo compartido

Es un sistema de trabajo de un computador gracias al cual se satisfacen los requerimientos de ejecución de diferentes programas de forma aparentemente simultánea. Esta ilusión es creada asignándole a cada programa pequeños momentos (llamados normalmente *QUANTUM*) de uso del *CPU*. Esto le crea al usuario la sensación de que cada programa se ejecuta en forma continua. Internamente el procesador alterna su dedicación entre los distintos programas pendientes.

Los Sistemas de Operación de tiempo compartido utilizan la planificación del procesador para dotar a cada usuario de una pequeña parte de los recursos del ordenador compartido.

Planificador

El **planificador** es un componente funcional muy importante de los sistemas de operación. Éste debe repartir el tiempo disponible de un procesador para la ejecución de todos los programas.

Instrumentación de aplicaciones

Es un proceso que permite al compilador, o herramientas específicas, agregar o modificar instrucciones de un programa fuente o incluso a programas ejecutables, previamente compilados y enlazados. Esto tiende a hacerse sin alterar la semántica de la actividad realizada por el programa. Esta instrumentación se realiza con el objetivo de obtener ciertos valores asociados a la ejecución de los programas: eficiencia, accesos a memoria, número de llamadas al sistema, etc.

Enunciado

Se desea que usted implemente un instrumentador de código ensamblado y un manejador de excepciones, en el archivo exceptions.s, en las secciones de texto de usuario (demarcada por la directiva .text) y de kernel (demarcada por la directiva .ktext) respectivamente.

Su programa tendrá como entrada un código ensamblado que contendrá varios segmentos de código, cada uno de los cuales será tratado como programas independientes. Su proyecto debe realizar dos tareas:

- El instrumentador debe tomar el código ensamblado de varios *programas*, recibidos desde el programa principal, y agregarles ciertas modificaciones.
- El manejador de excepciones debe simular el funcionamiento de un planificador de tiempo compartido en base a estos *programas* ya instrumentados.

Sobre el instrumentador

Como primera tarea se debe analizar el código de los programas y agregar a cada uno algunas instrucciones. Note que esto debe ser realizado para cada uno de los programas recibidos. La actividad que se requiere realizar es la de contabilizar el número de instrucciones ***add*** que cada programa realice a lo largo de su ejecución.

Esto debe hacerse generando excepciones después de la ejecución de cada instrucción **add**. En otras palabras, se debe insertar en el código ensamblado de cada programa una instrucción **break 0x20** justo después de cada instrucción **add.** Note que las instrucciones que originalmente estaban en el programa, después del **add,** deben desplazarse una palabra. En la siguiente tabla se ilustra este proceso, que por simplicidad se muestra sin ensamblar:

Programa original	**Programa Instrumentado**

sub $t0, $t3, $v0
add $s3, $t4, $a1
ori $t0, $t3, $v0
addu $sp $sp, $fp
addi $sp $sp, 34
add $sp $sp, $fp
li $t3 45
	sub $t0, $t3, $v0
	add $s3, $t4, $a1
	break 0x20
	ori $t0, $t3, $v0
	addu $sp $sp, $fp
	addi $sp $sp, 34
	add $sp $sp, $fp
	break 0x20
	li $t3 45

Al ejecutarse una instrucción break el sistema suspenderá la ejecución del programa principal y pasará a ejecutar la rutina manejadora de interrupciones de Mars, donde se contabilizará que hubo una operación **add** en el programa que estaba ejecutándose (el programa al que pertenece el **break 0x20** correspondiente).

Sobre los programas

Los programas serán definidos dentro de un archivo como el suministrado **myprogs.s**. Este archivo colocará en el arreglo PROGS la dirección de comienzo de cada uno de estos programas. De manera que el programa i comienza en la dirección de memoria establecida en la posición i del arreglo. Y termina al invocar el siguiente syscall con código 10. En el arreglo contendrá NUM_PROGS programas.

Estos programas pueden hacer uso de instrucciones de salto cortos. Sin enbargo, por simplicidad, para este proyecto, sólo se permitirá el **beq**. Los saltos serán a direcciones dentro de cada programa, es decir, puede asumir que no se hará saltos de un programa al área de otro programa. Note que esto debe considerarse al agregar las instrumentaciones ya que no hacerlo puede cambiar la semántica de los programas haciendo que ellos fallen. En otras palabras, note que la existencia de

estos beq, requiere un trato especial al momento de instrumentar el código. Considere la siguientes situaciones:

Caso 1	**Caso 2**	**Caso 3**	**Caso 4**
X: add beq $zero $t2 X	beq $zero $t2 X add X:	add X: beq $zero $t2 X	add beq $zero $t2 X X:

En los casos 3 y 4 no hay problemas al hacer la instrumentación. En los casos 1 y 2, el insertar la instrucción *break* después de la instrucción **add** genera un error en el programa ya que hace obsoleto el valor inmediato asociado a la dirección de salto.

Para este proyecto puede asumir que el caso 2 no va a suceder. Los casos 1, 3 y 4 si pueden presentarse en los programas que se deban instrumentar. Ud debe hacer las actualizaciones al ensamblaje de la instrucción **beq** de manera que actualice el valor de salto en forma apropiada.

También puede asumir que hay espacio suficiente después de cada programa que permita desplazar al menos las instrucciones que se requieran desplazar. Por ejemplo, si un programa Pi tiene 3 instrucciones **add** (se requiere agregar tres instrucciones break) así que después de su finalización habrá espacio para al menos tres instrucciones. Esto garantiza que el programa puede crecer hasta tres instrucciones, pudiendo agregar los tres break requeridos en los lugares pertinentes.

Note que todos los programas terminan con la secuencia que permite ejecutar el syscall con código 10. Uds no puede permitir que esta secuencia se ejecute debido a que terminaría ejecución completa. Ud debe reemplazar esa secuencia por la instrucción **break 0x10**.

La instrucción break

Esta instrucción genera una excepción, se puede usar diferentes códigos de manera de diferenciar ciertos operaciones diferentes. Por ejemplo, como en este proyecto, el código 0x10 se refiere a la finalización de uno de los programas y el código 0x20 se refiere a que se acaba de ejecutar una instrucción **add**. La única manera de determinar ese código es extrayéndolo del ensamblaje de la misma instrucción break. A continuación se muestra cómo se ensambla esta instrucción. Note que no se ajusta a los formatos vistos en clase.

Break			
break code	0	code	0xd
	6 bits	20 bits	6 bits
Cause a breakpoint exception. Exception 1 is reserved for the debugger			

Sobre el planificador

Un planificador real debe asignar una cantidad igual de tiempo a cada uno de los procesos. Sin embargo, Mars no cuenta con interupción de reloj. Por consiguiente, en este proyecto se utilizará interrupciones de teclado para realizar el cambio entre programas.

Cada vez que se presione la tecla 's', el control de CPU se le asignará al siguiente programa. Este debe continuar en el punto donde haya quedado pendiente la vez anterior que haya estado en ejecución. Si se presiona la tecla 'p', se debe pasar el control del CPU al programa anterior (según el orden en el arreglo PROGS). Antes de darle el control al nuevo programa, el planificador debe guardar en una estructura propia los contenidos de todos los registros de propósito general (con excepción de k0 y k1) para poder restaurarlos cuando se vuelva a ejecutar el programa saliente de forma tal de salvaguardar todo su ambiente de ejecución (registros).

Si un programa termina su ejecución no volverá a ser considerado para darle control del CPU. Un programa tendrá como última instrucción ***break 0x10***. Se debe imprimir un mensaje:

El programa i ha terminado su ejecución.

Note que un programa puede ejecutar la instrucción *break 0x20* (u otro número diferente al 0x10) y esto no debe ser considerado como el fin de la ejecución del programa.

También se desea que su manejador pueda detectar cuando se presione la tecla Esc. En este caso se debe detener la máquina, es decir, será el equivalente a ejecutar la secuencia:

```
li $v0 10
syscall
```

En este caso se debe imprimir un mensaje:

La maquina ha sido apagada. Status de los programas:

Programa i (Finalizado)

Numero de add: 20

Programa j (No Finalizado)

Numero de add: 20

Programa k (Finalizado)

Numero de add: 20

Cualquier tecla diferente a p, s o ESC debe ser ignorada.

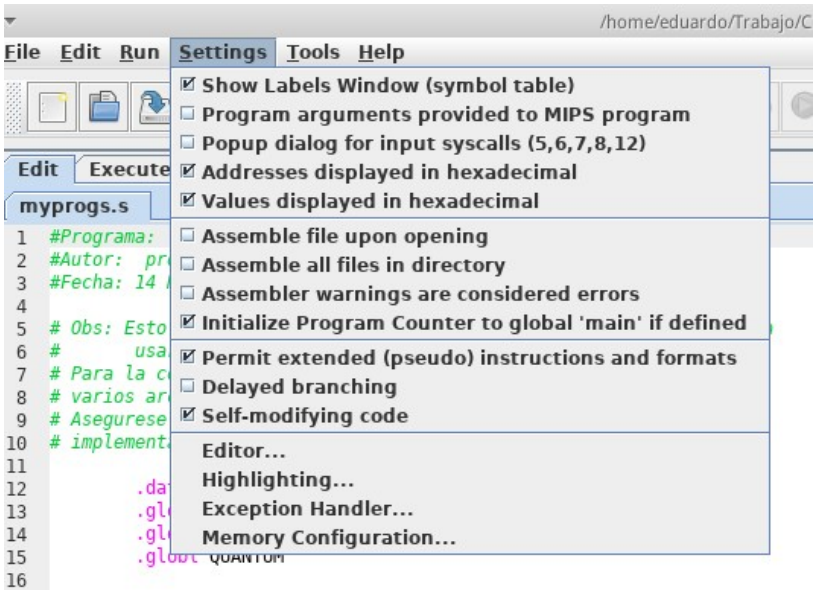
Recuerde que su manejador debe definir una serie de estructuras de datos para mantener información de todos los programas que se pueden ejecutar en el computador.

Los mensajes por defecto para las interrupciones manejadas deben ser suprimidos o sustituidos por los indicados en este enunciado.

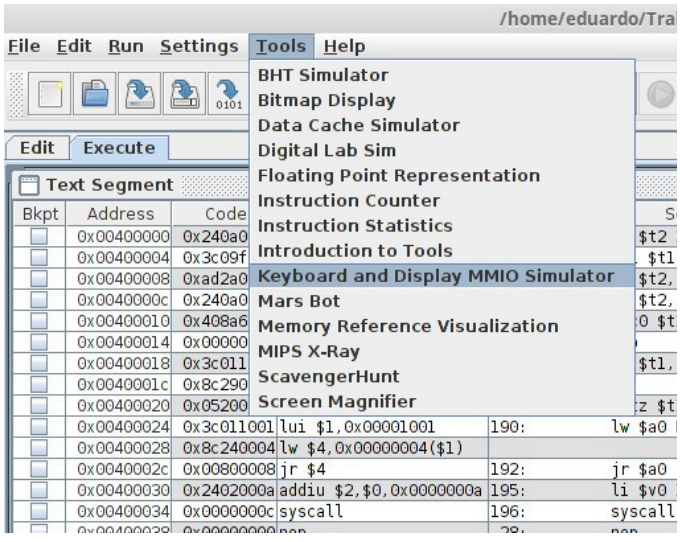
Detalles de implementación:

- El número de programas, que deben ejecutarse en tiempo compartido, estará almacenado en dirección asociada a la etiqueta **NUM_PROGS**. Definido en el programa principal, similar a como se hace en el archivo *myprogs.s* suministrado.

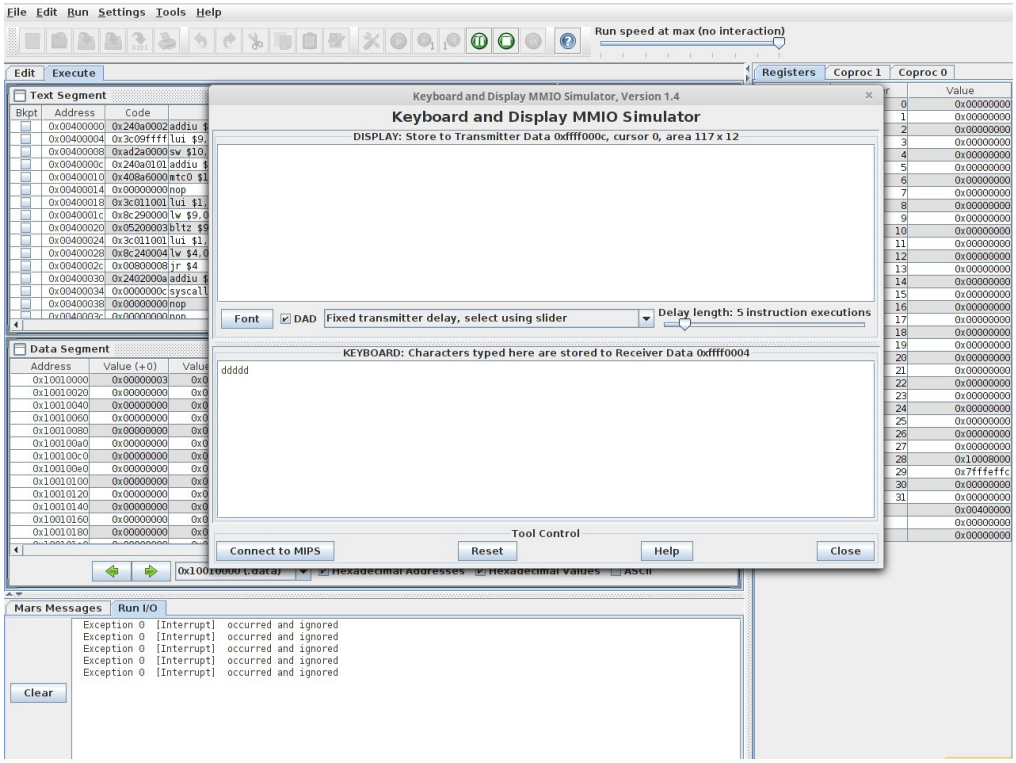
- Las direcciones de comienzo de los programas estarán almacenadas en el arreglo cuyo comienzo está almacenado en el arreglo referenciado por la etiqueta **PROGS**. El programa cuya dirección esté en la posición **PROGS**(X), con X = 4i y 0 <= i < **NUM_PROGS**, será conocido y referenciado como el programa i.
- El orden de ejecución de los programas va a estar determinado por el índice que éstos tengan asociado. El siguiente programa al programa i =N -1 es el programa i = 0.
- Asegúrese que la configuración de Mars es como lo muestra la siguiente figura:



Mars implementa los dispositivos como “Tools”. La ventana correspondiente al teclado se abre como se indica en la siguiente figura.



Una vez abierta, la ventana quedará sobre el manejador. Si desea ver lo que el programa imprime, debe colocar esta ventana de manera que no cubra el área de impresión (RUN I/O).



Material:

1. myexceptions.s a usar como base para su proyecto.
2. Un programa de pruebas (mymains.s) que muestra un posible archivo con 3 programas de prueba. Ud. DEBE escribir programas de pruebas que generen las interrupciones y excepciones que considere conveniente. Las etiquetas mencionadas, así como los diferentes programas, vendrán definidas en un programa como el presentado. No cambie los nombres.

El proyecto deberá ser entregado:

Lugar: en la oficina de su profesor de taller:

Hora: 7:30 am del viernes de la semana 11 sin prórroga. La corrida será realizada con los integrantes del grupo presente y tendrá una duración de un máximo de 15 minutos.

Sobre la Entrega

La entrega del programa es para el día jueves de la semana 11 antes de las 11:59pm. Ud deberá colocar su proyecto en Aula Virtual.

Adicionalmente, el día viernes Ud. debe entregar el código fuente impreso identificado con los nombres de los integrantes del grupo. El código debe tener una cantidad adecuada de comentarios; dentro de los comentarios debe estar bien clara la planificación de los registros.

No olvide entregarun reporte indicando las estructuras de datos definidas así como otras acciones tomadas.

El proyecto será probado con un archivo myprogs.s, desarrollado por el grupo profesoral. Su proyecto *debe* funcionar con dicho archivo.